Copy No. _____

Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

FEDEF: A High Level Architecture Federate Development Framework

James W. van Spengen

Defence R&D Canada – Atlantic

Technical Memorandum
DRDC Atlantic TM 2010-105
September 2010

Canada

This page intentionally left blank.

FEDEF: A High Level Architecture Federate Development Framework

James W van Spengen

Defence R&D Canada – Atlantic

Technical Memorandum
DRDC Atlantic TM 2010-105
September 2010

Principal Author

Original signed by James W. Van Spengen

James W van Spengen

Scientific Programmer

Approved by

Original signed by N.G. Pegg

N.G. Pegg

Head, Warship Performance

Approved for release by

Original signed by Ron Kuwahara for

C. Hyatt

Chair Defence Research Publications

Abstract

The development of High Level Architecture (HLA) federates can be repetitive and time consuming, with development time better spent on the aspects that differentiate the functionality provided by federates. The common tasks and design patterns involved in developing federates have been factored into an HLA federate development framework allowing for an easier and shorter federation development life cycle.

Normally, federate interoperability is limited to a specific run time infrastructure (RTI) vendor without recompilation of source code and is further constrained to a single HLA specification. This new framework aims to allow federate interoperability without the need to recompile source code when switching RTI vendors, and does not require code changes for operability between HLA specifications.

Configuration of federate requirements such as publications, subscriptions, time management, and management protocol should occur outside of federate source code, allowing for federate reusability without code modification and re-compilation. This HLA framework provides for single point external configuration of federates using an XML format.

Résumé

Le développement de fédérés d'architecture de haut niveau (HLA – High Level Architecture) est une tâche répétitive et longue. Il serait plus profitable d'utiliser ce temps pour définir les facteurs qui singularisent les fonctions fournies par les fédérés. Les tâches et les modèles de conception communs qui interviennent dans le développement de fédérés ont été intégrés dans un cadre de développement de fédérés HLA qui simplifie et accélère le cycle de développement de fédération.

En général, l'interopérabilité en matière de fédération est limitée à un seul fournisseur d'infrastructure valorisée à l'exécution (IVE) sans recompilation du code source et est de plus restreinte à une seule spécification HLA. Le nouveau cadre vise à permettre l'interopérabilité en matière de fédération sans qu'il soit nécessaire de recompiler le code source lorsqu'un fournisseur IVE est substitué à un autre ou de modifier le code pour assurer le fonctionnement sous diverses spécifications HLA.

La configuration des exigences en matière de fédération (publications, abonnements, gestion du temps, protocole de gestion, etc.) devrait s'effectuer à l'extérieur du code source, permettant aussi de réutiliser la fédération sans qu'il soit nécessaire de modifier et de recompiler le code. Le cadre HLA proposé prévoit l'utilisation d'un seul élément de configuration externe des fédérés dans un format XML.

This page intentionally left blank.

Executive summary

FEDEF: A High Level Architecture Federate Development Framework

James W van Spengen; DRDC Atlantic TM 2010-105; Defence R&D Canada – Atlantic; September 2010.

Introduction or background:

The High Level Architecture (HLA) is a framework for building complex distributed simulations referred to as federations. DRDC Atlantic is currently developing federations for simulating replenishment at sea and small boat launching and recovery in its Virtual Ship (VShip) Laboratory. A federation typically consists of multiple federates, (such as helm control, sea keeping, propulsions etc.), with each federate being an executable program that interacts with other federates using HLA standards and rules specific to the federation. Although HLA facilitates the development of complex simulations, it also introduces significant overheads for the development of software. These overheads are compounded by the existence of multiple application programme interfaces (APIs) for HLA run-time infrastructures (RTIs).

Results:

This document presents the FEDEF high level framework for developing HLA federates. The framework includes an API that can support the DMSO 1.3 and IEEE 1516 HLA standards, and the Mak, Pitch, and Portico RTIs. The framework also simplifies many programming tasks that are normally required when developing a federate. The framework currently supports time-stepped federations.

Significance:

The FEDEF framework enhances productivity and reduces errors when developing HLA federates. Developed federates can be re-configured for different RTIs without recompilation of source code. Developed federates are easier to maintain due to simplified source code.

Future plans:

The FEDEF framework will be extended to support additional HLA standards and RTIs. Future development will also consider support for event-driven federations.

Sommaire

FEDEF: A High Level Architecture Federate Development Framework

James W van Spengen; DRDC Atlantic TM 2010-105; R & D pour la défense Canada – Atlantique; Septembre 2010.

Introduction ou contexte :

L'architecture de haut niveau (HLA – High Level Architecture) est un cadre qui permet d'élaborer des simulations réparties complexes, nommées fédérations. RDDC Atlantique développe actuellement des fédérations pour simuler le ravitaillement en mer et la mise à l'eau et la récupération de petites embarcations dans son laboratoire de navires virtuels (VShip). Une fédération comprend normalement plusieurs fédérés (p. ex. : commande du gouvernail, tenue en mer, propulsion), des programmes qui interagissent entre eux par l'intermédiaire des normes et des règles HLA particulières de la fédération. L'utilisation de l'HLA simplifie le développement de simulations complexes, mais impose aussi des coûts de développement logiciel importants. L'existence de multiples interfaces de développement d'applications (API) pour les infrastructures valorisées à l'exécution (IVE) HLA ne fait qu'augmenter ces coûts.

Résultats :

Le document présente le cadre de haut niveau FEDEF qui permet de développer les fédérés HLA. Le cadre comprend une API qui prend en charge les normes DMSO 1.3 et IEEE 1516 et les IVE Mak, Pitch et Portico. Le cadre simplifie aussi de nombreuses tâches de programmation qui font normalement partie du développement d'un fédéré. Le cadre prend actuellement en charge les fédérations en temps réel.

Importance :

Le cadre FEDEF améliore la productivité et réduit les erreurs dans le développement de fédérés HLA. Une fois développés, les fédérés peuvent être reconfigurés pour diverses IVE sans recompilation du code source. Le code source simplifié facilite la maintenance des fédérés.

Perspectives :

Le cadre FEDEF sera élargi pour prendre en charge d'autres normes HLA et d'autres IVE. On envisage également le développement futur de fédérés capables de prendre en charge les fédérations fondées sur les événements.

Table of contents

List of figures

Acknowledgements

I wish to acknowledge my colleagues at the Simulation of Naval Platforms Laboratory, DRDC Atlantic, David Heath, James Nickerson and Shawn Oakey for their contributions to the Hierarchical Object Orientated Development (HOOD) toolkit, which was used extensively in the development of the HLA development framework.

This page intentionally left blank.

1 Introduction

The High Level Architecture (HLA) is a framework for building complex distributed simulations referred to as federations. A federation typically consists of multiple federates, with each federate being an executable program that interacts with other federates using HLA standards and rules specific to the federation. Although HLA facilitates the development of complex simulations, it also introduces significant overheads for the development of software. These overheads are compounded by the existence of multiple application programmer interfaces (APIs) for HLA run-time infrastructures (RTIs) which include the IEEE 1516 [1] and DMSO 1.3 [2] specifications.

The HLA 1516 and 1.3 specifications aim to provide a common interface for interacting with the run time infrastructure (RTI) for service requests and handling of call backs for federates communicating within a federation. Unfortunately in the context of using C++ namespaces and some differences in data types used in interface method signatures, it is not possible to write source code once for a federate and have it able to interact with all vendor RTI implementations. Further, it is difficult to integrate federates written to conform to the older HLA 1.3 specification with federates developed to the more recent HLA 1516 specification.

At the Simulation of Naval Platforms Laboratory, Defence Research and Development Canada (DRDC) - Atlantic, federates are usually provided by different contractors and must be integrated with other existing federates to create federations which are usually written for disparate RTI vendors and even differing RTI specification versions. Federate management patterns are also very different in areas such as synchronization methods for start up and shut down.

These technical constraints motivated the creation of an HLA federate development framework which would handle all the low level and highly repetitive configuration and management tasks required by most federates. Federates which utilize this framework are also free to interact with each other in different federations regardless of RTI vendor, specification version or management scheme.

The configuration of federate requirements with respect to the RTI, such as which objects to publish and subscribe, are often written into the federate source code and therefore unchangeable unless access to the source code is available for modification and recompilation. By factoring out configuration parameters to a single point Extensible Mark-up Language (XML) [3] file, changes to federate requirements can be easily made. Example excerpts of configuration file entries are included throughout this document where appropriate to help illustrate it's usage and context within the framework.

This document is not a detailed user manual for the framework but is intended to provide a higher level view to show what the framework is capable of and how it can simplify federate development.

2 Agile Framework Interface

2.1 HLA Ambassador Adapter

In order to create a framework designed to handle service requests and call backs with different RTI implementations and specifications, it was necessary to define a class which is essentially an incarnation of the adapter design pattern [4]. The adapter pattern converts an existing RTI vendor interface into a more generic interface that all federate clients can utilize and allows clients to collaborate within different federations. This class is called the HLAAmbassador and one instance is required for each federate. The HLAAmbassador provides one level of abstraction above the specific RTI vendor and relies on object composition to forward federate service requests and call backs between the client and the RTI vendor. The advantage to this approach is the federate client code is written to conform to a single HLAAmbassador interface.

In conventional federate design, a separate object is created and registered with the RTI to handle call backs from the RTI, while another object contains a pointer to an ambassador class through which service calls to the RTI can be made. This HLA framework combines both the RTI call backs and the ambassador pointer to the RTI in the single HLAAmbassador class, which simplifies the design by reducing the need for communication and synchronization between two separate objects. Therefore, a federate client need only use one HLAAmbassador object to meet all the federate RTI communication requirements.

Figure 1 shows the Unified Modeling Language (UML) class diagram [5] for the HLAAmbassador inheritance hierarchy for the 1516 specification. An identical hierarchy exists for the 1.3 specification. Each grandchild branch of HLAAmbassador is compiled into a distinct dynamically linked library (DLL), i.e. HLAAmbassadorMAK1516.dll. The choice of RTI vendor to use for the federation under development is made by using one of the compiled HLAAmbassador DLLs. In our current usage of the framework, the DLL is loaded dynamically at runtime, with the choice of DLL indicated in the federate XML configuration file, as demonstrated in Figure 2. Static linkage with the chosen vendor ambassador is also possible for binding at compile time.

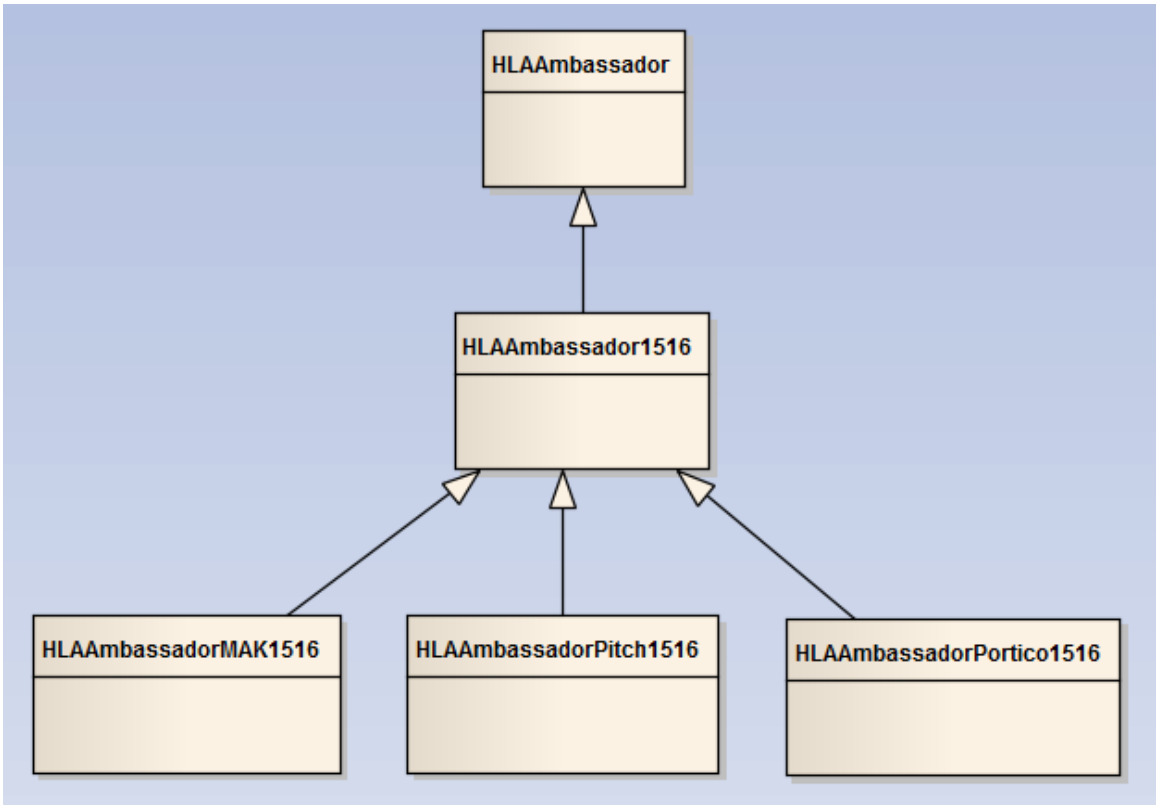

Figure 1 HLA Ambassador Class Hierarchy for 1516 Specification

```
<Federate>
        <RequiredRTI>HLAAmbassadorMAK1516d.dll</RequiredRTI>
</Federate>
```

Figure 2 XML Configuration Entry for Required RTI Vendor DLL

2.2 Observer Pattern Usage for Framework Updates

In designing the framework it was desirable to loosely couple the framework to its clients. In other words, the framework knows nothing about the federate clients that use its services. This

approach allows the framework and client code to be modified independently and encourages federate client code reuse; however, loose coupling poses challenges in communication between the framework and its clients. To overcome this obstacle and still maintain the design goals, the observer design pattern [4] is used, as shown schematically in Figure 3.

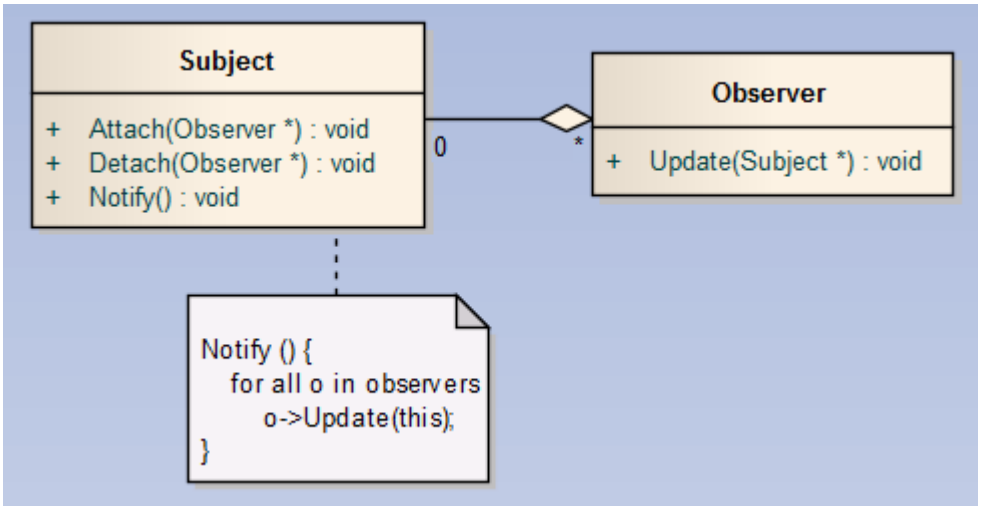

Figure 3 Observer Pattern

The observer pattern creates a one-to-many relationship between the framework and its clients so that when the single HLAAmbassador receives state changes from the federation, it is communicated to any number of objects within the federate client. Unlike conventional federate design, the service call backs from the RTI are not constrained to be received by the object designated as the federate ambassador, but can be received by any objects in the federate client which implement the appropriate interface. On reception of the state change, the client is able to query the subject being observed and obtain the state data.

The state changes of interest that are made available through the framework include object attribute updates, interaction reception, object management, and synchronization point announcements. Federate clients receive state change updates by inheriting from one or more interfaces which provide the call back method signatures used by the HLAAmbassador. Clients must also indicate their interest in state changes by registering themselves with the HLAAmbassador.

2.3 HLA Meta Data Encapsulation With FOM Entity Classes

Data entities, which are made available among federates, are specified in the Federate Object Model (FOM) [1] [6]. FOM data entities are communicated via the RTI. Typical federate implementations contain a wide variety and large volume of data obtained from the RTI, which the developer needs to store, make available for use by different objects, and keep consistent with the current state of the RTI. To alleviate the developer of this burden, the framework encapsulates the HLA meta data within classes which represent the FOM entities being modelled. The HLAAmbassador automatically retrieves the data from the RTI, populates the required FOM

entities and makes the data available to the federate client through intuitive class models, as illustrated in Figures 4, 5, and 6.

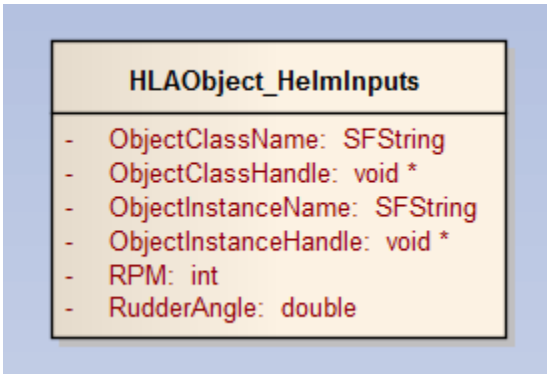

Figure 4 Entity Class Representation of HelmInputs FOM Object

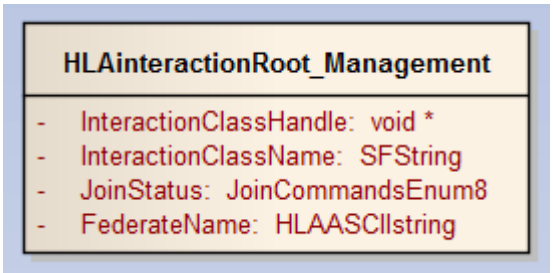

Figure 5 Entity Class Representation of Management FOM Interaction

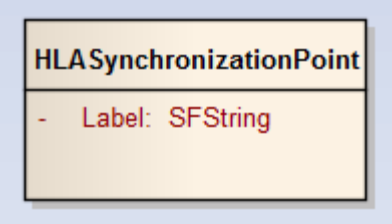

Figure 6 Entity Class Representation of Synchronization Point

FOM entity objects are the primary means of communication between the HLAAmbassador and client code and contain entity related data such as instance name, instance handle, class handle, class name and, most importantly, attribute data to send or receive from the RTI. An entity class exists for each object, interaction and data type as detailed in the FOM used by the federation. Also included are entity classes for synchronization points which are usually required for management of the federation.

FOM entity objects automatically handle encoding/marshalling and decoding/un-marshalling of data into the format required for sending and receiving using RTI services. This includes detecting and accounting for machine endianess.

A typical federation FOM can contain numerous objects, interactions and data types. To eliminate the need for the developer to create all these classes, a code generation tool has been developed which parses an XML FOM document and generates the required classes. The code generation tool does not support creation of classes from HLA 1.3 FED [7] files.

2.4 Federate Initialization and Federation Joining

The initialization of federates, which configures them with the required parameters to join the federation and obtains all the meta data they need from the RTI, involves the repetition of numerous tasks. A typical federate configuration involves the following procedures:

- querying for federation existence.
- creating federation execution.
- joining federation execution.
- retrieving handles for objects, attributes, interactions, and parameters.
- registering interactions, object publications, and subscriptions.
- requesting unique names for publication.

Using this framework liberates the developer from these initialization tasks, which are handled automatically. Configuration parameters required by the RTI are indicated in the federate XML configuration file, as illustrated in Figure 7, and are made available to the framework once parsed by the HLAAmbassador.

```
<Federate>
   <RequiredRTI>HLAAmbassadorMAK1516d.dll</RequiredRTI>
   <FederateManager>HLAManagerDRDCd.dll</FederateManager>
   <FederationName>DRDC Seaway Federation</FederationName>
   <RTIConfigurationFile>%HOODTK%\CommonData\HLAData\RIDs\
      rid_MAK_v3.4.mtl</RTIConfigurationFile>
   <FOM>%HOOD_DIR%\CommonData\HLAData\FOMs\
      Ship_Seaway_FOM.xml</FOM>
   <FederateType>Ship Motion</FederateType>
</Federate>
```

Figure 7 XML Configuration Entry for Federate Initialization Parameters

After parsing the XML configuration file, the developer calls the initialize method on the HLAAmbassador to have the framework perform the following:

- creation of the federation execution if necessary.

- join the federate to the federation execution.
- obtain all required entity handles from the RTI.
- configure publication and subscription of objects and interactions
- register unique object names for publication.

A code example of the initialization method calls is given in Figure 8.

```
HoodXML::XQillaHelper xQilla;

xQilla.parseXMLFile( ( const wchar_t * )configFileName );

m_federate->parseConfigurationFile( xQilla );

m_federate->initialize();
```

Figure 8 Federate Configuration and Initialization Tasks

2.5 Object Attribute Publication and Subscription

At the heart of communication between federates within a federation is the notion of object attributes. FOM entities, which usually model a real world concept, are designated in the FOM as objects and contain one or more attributes. Attributes represent and contain the data which describe the object being modelled and are the lowest denominator for specifying which object data are transmitted within the federation.

Within this framework, attribute subscriptions and publications are specified within the federate configuration file instead of within the federate code. An example object publication excerpt is shown in Figure 9. Once configured, the framework is responsible for creating and registering the objects and attributes with the RTI and making them available to the federate client for manipulation.

```
<FederateObjectPublications>
  <Publication>
    <ObjectClass>HLAobjectRoot.BaseEntity.PhysicalEntity.Platform.Surface
    Vessel</ObjectClass>
    <InstanceName>Ship</InstanceName>
    <Attributes>
      <Attribute>Spatial</Attribute>
      <Attribute>ShipRpm</Attribute>
      <Attribute>ShipHeading</Attribute>
      <Attribute>ShipRudderAngle</Attribute>
      <Attribute>ShipSpeed</Attribute>
    </Attributes>
  </Publication>
</FederateObjectPublications>
```

Figure 9 XML Configuration Entry for Object Publication

For attribute publication by a federate, the client obtains a pointer to the desired object to publish from the HLAAmbassador, as shown in Figure 10.

```
HLAAmbassador * m_federate;
/*
* Setup the surface vessel object pointer to be used for
* attribute publication
*/
m_surfaceVessel =
      dynamic_cast< HLAobjectRoot_BaseEntity_PhysicalEntity_Platform
      _SurfaceVessel *>( m_federate->getPublishedObject( L"Ship" ) );
```

Figure 10 Client Obtaining HLA Object Pointer for Attribute Publication

Notice that the instance name of the HLA object to be published is passed to the HLAAmbassador which then retrieves a pointer to the desired instance object. Once retrieved, the HLA object's attribute values can be set and then sent for publication to the federation, as demonstrated in Figure 11.

```
m_surfaceVessel->setShipRudderAngle( rudderValuesArray );

m_federate->publishAttributeValues( m_surfaceVessel,
                                    VariableLengthData(),
                                    p_requestTime );
```

Figure 11 Setting HLA Object Attribute Values and Publication

The framework supports publication of attribute values both with and without a time stamp value.

The receiving of attribute values published to the federation is accomplished through an interface called IAttributeObserver, outlined in Figure 12, which contains call backs made from the HLAAmbassador passing the HLA object with updated attribute values to the client. Any client object wanting to receive attribute updates inherits from this interface overriding the call backs.

```
virtual void attributeUpdate ( const HLAObject * p_object,
                               const VariableLengthData &
                                     p_userSuppliedTag );

virtual void attributeUpdate ( const HLAObject * p_object,
                               const VariableLengthData &
                                     p_userSuppliedTag,
                               double p_time );
```

Figure 12 IAttributeObserver Interface

Objects indicate their interest in an attribute update by registering themselves with the HLAAmbassador, passing in a pointer to the object to receive the updates and the instance name of the HLA object from which to receive updates. An example of this method call is given in Figure 13. An object designated as receiving the attribute updates can be used to receive updates on any number of HLA objects. Updates from a single HLA object can be sent to more than one client object.

```
m_federate->registerAttributeSubscriptionObserver( this, L"Helm" );
```

Figure 13 Registering a Client to Receive Attribute Update Call Backs

2.6 Interaction Publication and Subscription

Interactions within a federation are much like objects in that they are used to pass data of interest between federates. Interactions, unlike objects, are usually not published with each federation time step and only occur occasionally. They are useful for modelling events occurring within the federation and contain one or more parameters. Parameters represent and contain the data which describe the event being modelled.

Within this framework, interaction subscriptions and publications are specified within the federate configuration file instead of within the client code. An example interaction subscription excerpt is shown in Figure 14. Once configured, the framework is responsible for creating and registering the interactions and parameters with the RTI and making them available to the federate client for manipulation.

```
<FederateInteractionSubscriptions>
  <Subscription>
    <InteractionClass>HLAinteractionRoot.Management</InteractionClass>
    <Parameters>
      <Parameter>JoinStatus</Parameter>
      <Parameter>FedName</Parameter>
    </Parameters>
  </Subscription>
</FederateInteractionSubscriptions>
```

Figure 14 XML Configuration Entry for Interaction Subscription

The framework patterns for publishing and subscribing interactions are analogous to those used for object attributes. For interaction publication within a federate, the federate client obtains a pointer to the interaction instance to publish from the HLAAmbassador, as shown in Figure 15. Interactions do not have instance names so the type of interaction required is passed to the retrieval call.

```
m_managementInteraction =
      dynamic_cast< HLAinteractionRoot_Management * > (
      m_federate->getPublishedInteraction( L"Management" ) );
```

Figure 15 Client Obtaining Interaction Pointer for Publication

Once retrieved, the HLA interaction's parameter values can be set and then sent for publication to the federation, as shown in Figure 16. The framework supports publication of interactions both with and without time stamps.

```
m_managementInteraction->setJoinStatus( AllQuit );

m_federate->sendInteraction( m_managementInteraction,
                             VariableLengthData() );
```

Figure 16 Setting Interaction Parameter Values and Publication

The receiving of interaction values published to the federation is accomplished through an interface called IInteractionObserver, outlined in Figure 17, which contains the call backs made from HLAAmbassador passing the interaction object with updated values to the client. Any client wanting to receive interaction updates inherits from this interface overriding the call

backs.

```
virtual void interactionUpdate ( const HLAInteraction * p_interaction,
                                 const VariableLengthData &
                                    p_userSuppliedTag );

virtual void interactionUpdate ( const HLAInteraction * p_interaction,
                                 const VariableLengthData &
                                     p_userSuppliedTag,
                                 double p_time );
```

Figure 17 IInteractionObserver Interface

Client objects indicate their interest in an interaction update by registering themselves with the HLAAmbassador, passing in a pointer to the object to receive the updates and the type of instance of interaction from which to receive updates. An example of this method call is given in Figure 18. A client object designated as receiving the interaction updates can be used to receive updates from any number of interaction types. Conversely, updates from a single interaction can be sent to more than one client object.

```
m_federate->registerInteractionSubscriptionObserver( this,
      L"Management" );
```

Figure 18 Registering a Client to Receive Interaction Update Call Backs

2.7 Synchronization Point Management

Synchronization points are used within a federation to signal that a point has been reached within a sequence of events, and are identified with a string label. They are normally used to coordinate the start-up sequence of federates when launching a federation.

Synchronization points that are required by a federate are indicated in the federate XML configuration file. An example synchronization point configuration excerpt is given in Figure 19.

```
<SynchronizationPoints>
  <SynchronizationLabel>ReadyToRun</SynchronizationLabel>
 </SynchronizationPoints>
```

Figure 19 XML Configuration Entry for Synchronization Point

One federate within the federation, usually dedicated as the federation manager, must register the synchronization points with the RTI, as illustrated in Figure 20.

```
m_federate->registerSynchronizationPoints();
```

Figure 20 Registering Synchronization Points

Indication that a synchronization point has been reached within federate client code is achieved by calling an HLAAmbassador method, and passing in the identifying label of the synchronization point, as illustrated in Figure 21.

```
m_federate->synchronizationPointAchieved( L"ReadyToRun" );
```

Figure 21 Announcing Reaching Synchronization Point

The receiving of synchronization call backs from the RTI is accomplished through an interface called ISynchronizationPointObserver, outlined in Figure 22. Any client wanting to receive synchronization call backs inherits from this interface and overrides the methods.

```
virtual void announceSynchronizationPoint(
      const HLASynchronizationPoint * p_synchronizationPoint,
      const VariableLengthData & p_userSuppliedTag );

virtual void synchronizationPointRegistrationSucceeded(
      const HLASynchronizationPoint * p_synchronizationPoint );

virtual void synchronizationPointRegistrationFailed(
      const HLASynchronizationPoint * p_synchronizationPoint );

virtual void federationSynchronized(
      const HLASynchronizationPoint * p_synchronizationPoint );
```

Figure 22 ISynchronizationPointObserver Interface

Client objects indicate their interest in a synchronization call back by registering themselves with the HLAAmbassador, passing in a pointer to the object to receive the call backs and the label identifying the synchronization point from which to receive updates. An example of this method call is given in Figure 23. A client object designated as receiving the synchronization call backs can be used to receive updates from any number of synchronization labels. Conversely, call backs from a single synchronization label can be sent to more than one client object.

```
m_federate->registerSynchronizationPointObserver( this,
      L"ReadyToRun" );
```

Figure 23 Registering a Client Object to Receive Synchronization Point Call Backs

2.8 Object Management Services

During execution, it is sometimes useful for a federate to be aware of the life span of object entities that are made available by another federate for attribute updates within the federation. The HLA specifications provide mechanisms for managing the object within a federate client and these are mirrored in the framework.

Client federates can receive call backs from the HLAAmbassador by implementing the IObjectManagementObserver interface, overriding the call back methods which provide the

managed object as an input parameter to be queried by the client. The call back signatures are outlined in Figure 24. The provided HLAObject parameter will contain the object instance handle as given by the RTI, and its value is only applicable within the federate client. Call backs are also provided for notification when an object instance is removed from the federation.

```
virtual void objectDiscovered ( HLAObject * p_object );

virtual void removeObjectInstance ( const HLAObject * p_object,
      const VariableLengthData & p_userSuppliedTag, double p_time );

virtual void removeObjectInstance ( const HLAObject * p_object,
      const VariableLengthData & p_userSuppliedTag );
```

Figure 24 IObjectManagementObserver Interface Call Backs

Client objects indicate their interest in receiving object management call backs by registering themselves with the HLAAmbassador, passing in a pointer to the object to receive the call backs. An example of this method call is given in Figure 25. More than one client object can be designated to receive the call backs by repeating the register method call.

```
m_federate->registerObjectManagementObserver( this );
```

Figure 25 Registering a Client Object to Receive Object Management Call Backs

Keep in mind that the HLA specification constrains that object discovery can only be made with object types for which the federate client has expressed interest in receiving attribute updates.

2.9 Time Management

Federates operating within a federation can have several modes of operation, in regards to time management in the context of HLA simulations, which are time-stepped or event-driven [8]. This framework provides the mechanisms necessary for time-stepped time management only, reserving event-driven support for future work.

Time stepped federates are configured with two parameters called time-regulating and time-constrained, both of which contain boolean true or false values, creating a matrix of four possible modes [8] of time-stepped operation. These values are set for each federate through its external configuration file and are conveyed through to the RTI automatically by the framework. A look ahead value [9] guaranteeing no reception of updates below a certain time value is also indicated in the XML configuration file. An example excerpt of time configuration is given in Figure 26.

```
<Federate>
        <TimeConstrained>true</TimeConstrained>
        <TimeRegulating>true</TimeRegulating>
        <LookAhead>0.1</LookAhead>
</Federate>
```

Figure 26 XML Configuration Entry for Time Management Parameters

The framework includes several methods useful for controlling time-stepped federates such as requesting advances to the simulation time, requesting a pause in federate execution to allow for call backs to be received from the HLAAmbassador and finally the ability to change, during federation execution, the federate look ahead value. The available methods for time management are given in Figure 27.

```
virtual void timeAdvanceRequest ( double p_requestedTime,
                                  const bool & p_wait = true );

virtual void waitForCallbacks ( double p_minimumTime,
                                double p_maximumTime );

void setLookAhead ( double p_lookAhead );

const SFBool & getTimeConstrained () const;

const SFBool & getTimeRegulating () const;

const SFFloat & getSimulationTime () const;

const SFFloat & getLookAhead () const;
```

Figure 27 HLAAmbassador Methods Provided for Time Management Service Requests

Time management call backs provided by the HLA specification, such as timeAdvanceGrant, are not available to federate client code, as the framework handles these call backs for the client.

3 Federate Resignation and Federation Destruction

When federate execution completes, the framework provides services for resigning from and destroying the federation execution through the method call resign. Resign performs two actions:

1. resigns the federate from the federation.

2. destroys the federation execution if no other federates are joined to the federation.

Resignation from the federation implies that object instances owned by the federate are deleted from the federation and the associated attributes are not available for ownership transfer. The ability to change the resignation action in relation to owned object attributes is reserved for future work.

The call to resign automatically attempts to destroy the federation execution, but is only successful if no other federates are joined. An example of a federate calling for resignation is given in Figure 28.

```
if ( m_federate )
{
      m_federate->resign();
      delete m_federate;
      m_federate = NULL;
}
```

Figure 28 Invoke Framework to Resign Federate

4 A Model for Federate Lifecycle Management

As explained earlier, federates can be developed to different HLA specifications and RTI vendor implementations. This motivated the development of the HLA development framework. A similar problem exists in which federates can employ different strategies for federate lifecycle management to determine events such as when to begin time stepping and when to shut down. To follow the same philosophy of creating an adaptable and reusable federate life cycle management framework, a common interface was created for signalling the life cycle stages of a federate.

The life cycle management framework exists separately from the HLA federate development framework and its inclusion in federate development is optional.

The following sequence of life cycle stages, within which the federate exists, are implemented as follows:

1. Determination that all required federates have joined the federation.

2. Determination that all required federates have completed initialization tasks and are ready to begin time stepping.

3. Determination that a request has been made to resign federates and possibly destroy the federation execution.

The life cycle management interface resides in the base class FederateManager, see Figure 29, to be inherited for use in the development of specific management schemes. Federates can then be designed to manage their life cycle to this common interface and operate independently of any specific life cycle management strategies.

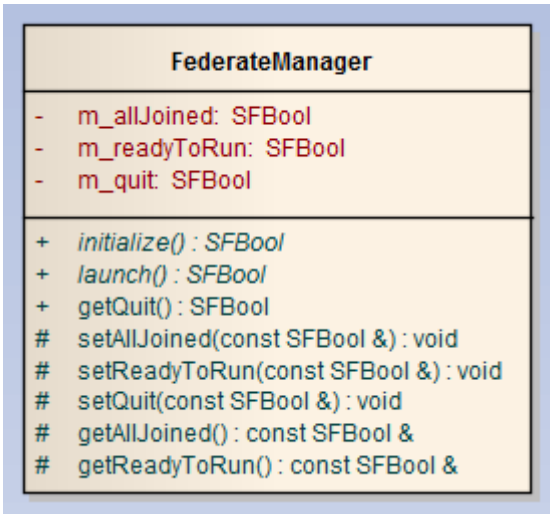

Figure 29 Class Diagram for the Federate Lifecycle Manager Interface

The core methods of the interface are two pure virtual methods, initialize and launch, which are meant to be overridden by the implementation. Initialize is intended to bring the federate through all federates joined and ready to begin time stepping stages of the lifecycle. Once initialize returns a value of true to the caller, the federate can begin time stepping. Determination of the signal for federate resignation is made by querying getQuit, at which point the federate will resign making a service call to the HLAAmbassador. An example of a federate calling a management object is given in Figure 30.

```
if ( !m_manager->launch() )
{
	cleanUp();
	return;
}

while ( !m_manager->getQuit() )
{
	double time = m_federate->getSimulationTime() + m_federate->
		getLookAhead();
	publishAttributes( time );
	m_federate->timeAdvanceRequest( time );
}
```

Figure 30 Example Usage of Federate Life Cycle Manager

5 Future Services Inclusion

It is recognised that not all services offered by the HLA specification have been implemented in the framework, and are intended to be included in future development including the following:

- Indicating interest in attribute updates through start and stop registration RTI call backs.
- Indicating interest in attribute updates for specific object instances through turn updates on and off RTI call backs.
- Indicating interest in interactions through on and off RTI call backs.
- The ability to un-publish object attributes and interactions.
- The ability to un-subscribe to object attributes and interactions.
- The ability to delete object instances.
- Switching of attribute ownership between federates.
- Mechanism to request and receive updates of attribute values for late joining federates.
- Add features to enable event driven federation advancement.

Other federation management models need to be explored and made available for use.

6 Concluding Remarks

The HLA development framework considerably reduces the amount of effort required to develop federates from scratch and to modify existing federates to fit the framework. It has allowed the creation of a large library of federates which are interoperable regardless of RTI vendor or HLA specification.

References

[1] IEEE 1516.1 (2000), Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification, IEEE.

[2] Modeling & Simulation Coordination Office (April 1998), HLA Interface Specification Version 1.3, Department of Defense, USA.

[3] Extensible Mark-up Language (XML), W3C, http://www.w3.org/XML/ (Access date: 14 July 2010).

[4] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (2005), *Design Patterns, Elements of Reusable Object-Orientated Software*, Addison-Wesley, pp. 139-150, 293-303.

[5] Booch, G., Rumbaugh, J. and Jacobson, I. (2001), *The Unified Modelling Language User Guide*, Addison-Wesley.

[6] IEEE 1516.2 (2000), Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT) Specification, IEEE.

[7] Department of Defense (1998), High Level Architecture Federation Execution Details (FED) File Specification, Department of Defense, USA.

[8] Kuhl, F., Weatherly, R. and Dahmann, J. (1999), *Creating Computer Simulation Systems*, Prentice Hall.

[9] Fujimoto, R. (2000), *Parallel and Distributed Simulation Systems*, John Wiley & Sons.

Annex A Public Interface for HLAAmbassador Class

```
virtual void parseConfigurationFile ( const XQillaHelper &
                                      p_xQillaHelper );

/*
* Synchronization Point Management
*/
virtual void registerSynchronizationPoints ();

virtual void registerSynchronizationPointObserver (
                        ISynchronizationPointObserver * p_observer,
                        const SFString & p_label );

virtual void synchronizationPointAchieved ( const SFString & p_label );

/*
* Synchronization Point Callbacks
*/
void synchronizationPointRegistrationSucceeded ( const SFString &
                                                 p_label );

void synchronizationPointRegistrationFailed ( const SFString &
                                              p_label );

void announceSynchronizationPoint ( const SFString & p_label,
                                    VariableLengthData & p_tagData );

void federationSynchronized ( const SFString & p_label );

/*
* Interaction Management
*/
virtual void registerValidDynamicInteractions( void( * functionPtr )(
                        ValidDynamicInteractionList & ) );

virtual void registerInteractionSubscriptionObserver (
                               IInteractionObserver * p_observer,
                               const SFString & p_interactionClassName );

virtual void sendInteraction ( const HLAInteraction * p_interaction,
                               const VariableLengthData & p_tagData );

virtual void sendInteraction ( const HLAInteraction * p_interaction,
                               const VariableLengthData & p_tagData,
                               double p_time );
```

```
/*
* Attribute Management
*/
virtual void registerAttributeSubscriptionObserver (
                              IAttributeObserver * p_observer,
                              const SFString & p_objectInstanceName );

virtual void publishAttributeValues ( const HLAObject * p_object,
                                 const VariableLengthData & p_tagData );

virtual void publishAttributeValues ( const HLAObject * p_object,
                                   const VariableLengthData & p_tagData,
                                   double p_time );

/*
* Object Management
*/
virtual void registerValidDynamicObjects ( void ( * functionPtr ) (
                                           ValidDynamicObjectList & ) );

virtual void registerObjectManagementObserver ( IDiscoverObjectObserver
                                                    * p_observer );

/*
* Time Management
*/
virtual void timeAdvanceRequest ( double p_requestedTime,
                                  const bool & p_wait = true );

virtual void waitForCallbacks ( double p_minimumTime,
                                double p_maximumTime );

void setLookAhead ( double p_lookAhead );

const SFBool & getTimeConstrained () const;

const SFBool & getTimeRegulating () const;

const SFFloat & getSimulationTime () const;

const SFFloat & getLookAhead () const;
```

```
/*
* Federate Life Cycle Management
*/
virtual bool initialize ();

virtual void resign ();

const SFString & getFederateName () const;

bool getTimeConstrainedEnabled () const;

bool getTimeRegulationEnabled () const;

bool getTimeAdvanceGranted ();

HLAObject * getPublishedObject ( const SFString & p_objectInstanceName
                                    );

HLAInteraction * getPublishedInteraction ( const SFString &
                                           p_intercationClassName );

const ObjectList & getObjectSubscriptions () const;
```

This page intentionally left blank.

Annex B Sample Federate XML Configuration

```
<?xml version="1.0" encoding="utf-8"?>
<ShipMotionFederate>

   <OutputFile log="false">ShipMotion_SupplyShip_Output.dat</OutputFile>

   <ShipMotion>
      <InputFile>./config/moFedDMSO2_Supply.inp</InputFile>
      <OutputFile>./config/moFedDMSO2_Supply.out</OutputFile>
   </ShipMotion>

   <Federate>

      <RequiredRTI>HLAAmbassadorMAK1516d.dll</RequiredRTI>
      <FederateManager>HLAManagerDRDCd.dll</FederateManager>
      <FederationName>DRDC Seaway Federation</FederationName>
      <RTIConfigurationFile>%HOODTK%\CommonData\HLAData\RIDs\
          rid_MAK_v3.4.mtl</RTIConfigurationFile>
      <FOM>%HOOD_DIR%\CommonData\HLAData\FOMs\Ship_Seaway_FOM.xml</FOM>
      <FederateName>Ship Motion</FederateName>
      <LookAhead>0.1</LookAhead>
      <TimeConstrained>true</TimeConstrained>
      <TimeRegulating>true</TimeRegulating>

      <SynchronizationPoints>
         <SynchronizationLabel>ReadyToRun</SynchronizationLabel>
      </SynchronizationPoints>

      <FederateObjectPublications>

         <Publication>
           <ObjectClass>HLAobjectRoot.BaseEntity.PhysicalEntity.Platform.
                  SurfaceVessel</ObjectClass>
           <InstanceName>Ship</InstanceName>
           <Attributes>
              <Attribute>Spatial</Attribute>
              <Attribute>ShipRpm</Attribute>
              <Attribute>ShipHeading</Attribute>
              <Attribute>ShipRudderAngle</Attribute>
              <Attribute>ShipSpeed</Attribute>
           </Attributes>
         </Publication>
      </FederateObjectPublications>
```

```
    <FederateObjectSubscriptions>
      <Subscription>
        <ObjectClass>HLAobjectRoot.HelmInputs</ObjectClass>
        <InstanceName>SUPPLY_HELM</InstanceName>
        <Attributes>
          <Attribute>helmAutoPilotHeading</Attribute>
          <Attribute>helmRpm</Attribute>
        </Attributes>
      </Subscription>
    </FederateObjectSubscriptions>

    <FederateInteractionPublications>
      <Publication>
        <InteractionClass>HLAinteractionRoot.Management</InteractionClass>
        <Parameters>
          <Parameter>JoinStatus</Parameter>
          <Parameter>FedName</Parameter>
        </Parameters>
      </Publication>
    </FederateInteractionPublications>

    <FederateInteractionSubscriptions>
      <Subscription>
        <InteractionClass>HLAinteractionRoot.Management</InteractionClass>
        <Parameters>
          <Parameter>JoinStatus</Parameter>
          <Parameter>FedName</Parameter>
        </Parameters>
      </Subscription>
    </FederateInteractionSubscriptions>

  </Federate>

</ShipMotionFederate>
```

List of symbols/abbreviations/acronyms/initialisms

DMSO	Defense Modeling and Simulation Office
DND	Department of National Defence
DRDC	Defence Research & Development Canada
FED	Federation Execution Data
FOM	Federation Object Model
HLA	High Level Architecture
IEEE	Institute of Electrical and Electronics Engineers
R&D	Research & Development
RTI	Run Time Infrastructure
XML	Extensible Mark-up Language

This page intentionally left blank.

Distribution list

Document No.: DRDC Atlantic TM 2010-105

LIST PART 1: Internal Distribution by Centre

3 Author (2 hardcopies and 1 CD)
3 DRDC Atlantic Library (1 hardcopy and 2 CDs)

6 TOTAL LIST PART 1

LIST PART 2: External Distribution by DRDKIM

1 Library and Archives Canada Atten: Military Archivist, Governments Records Branch
1 DRDKIM

2 TOTAL LIST PART 2

8 TOTAL COPIES REQUIRED

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence R&D Canada – Atlantic
9 Grove Street
P.O. Box 1012
Dartmouth, Nova Scotia B2Y 3Z7

2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.)

UNCLASSIFIED

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

FEDEF: A High Level Architecture Federate Development Framework

4. AUTHORS (last name, followed by initials – ranks, titles, etc. not to be used)

James W van Spengen

5. DATE OF PUBLICATION (Month and year of publication of document.)	6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.)	6b. NO. OF REFS (Total cited in document.)
September 2010	42	9

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Memorandum

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Defence R&D Canada – Atlantic
9 Grove Street
P.O. Box 1012
Dartmouth, Nova Scotia B2Y 3Z7

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)
11ge06	

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
DRDC Atlantic TM 2010-105	

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

Unlimited

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.))

Unlimited

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

The development of High Level Architecture (HLA) federates can be repetitive and time consuming, with development time better spent on the aspects that differentiate the functionality provided by federates. The common tasks and design patterns involved in developing federates have been factored into an HLA federate development framework allowing for an easier and shorter federation development life cycle.

Normally, federate interoperability is limited to a specific run time infrastructure (RTI) vendor without recompilation of source code and is further constrained to a single HLA specification. This new framework aims to allow federate interoperability without the need to recompile source code when switching RTI vendors, and does not require code changes for operability between HLA specifications.

Configuration of federate requirements such as publications, subscriptions, time management, and management protocol should occur outside of federate source code, allowing for federate reusability without code modification and re-compilation. This HLA framework provides for single point external configuration of federates using an XML format.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

HLA; Framework; Distributed Simulation; Simulation;

This page intentionally left blank.

Defence R&D Canada

Canada's leader in defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale

DEFENCE DÉFENSE

www.drdc-rddc.gc.ca